Vol.2012-HPC-135 No.24
2012/8/2

ワークフローアプリケーション基盤としての並列 DB の性能評価

中谷 翔[1,a)]　Ting Chen[1,b)]　田浦 健次朗[1,c)]

概要：大規模クラスタ環境やクラウドシステムのような計算資源で，巨大なデータを扱うデータ集約的なアプリケーションが増加傾向にある．このような処理を高効率に行うには， MapReduce 処理系が利用されることが主流であるが，データの統一的，高速な管理という観点からは，並列 RDBMS を基盤にしたワークフロー処理が有望であると考えられる．ワークフロー基盤として並列 RDBMS を利用する場合には，同一クエリの並列実行と，データベースから取得したデータに対する外部コマンドの透過的な適用という機構が重要であり，これらを兼ね備えた並列 RDBMS である ParaLite を提示する．本論文では，ParaLite の基本的な性能をオープンソースソフトウェアの並列 RDBMS である MySQL Cluster と比較した．また，ParaLite によって格フレーム解析ワークフローを一部動作させ，その性能について議論した．

キーワード：並列 RDBMS, ParaLite, MySQL Cluster, ワークフローアプリケーション

Evaluation of Parallel DB as Workflow Applications Framework

Sho Nakatani[1,a)]　Ting Chen[1,b)]　Kenjiro Taura[1,c)]

Abstract: Nowadays, more and more data-intensive applications which run on large cluster environments and cloud systems are developed. While MapReduce implementations are widely used for this end, Parallel RDBMS has the possibility to manage data in such applications with high efficiency and performance when it is used as a workflow applications framework. It is important for parallel RDBMS as a workflow applications framework to perform the same queries in parallel and execute external commands transparently. In this paper, we introduce ParaLite. We compared basic performance of ParaLite with MySQL Cluster. Also, we partially run Case Frame Construction workflow application with ParaLite and discussed on its performance.

Keywords: Parallel RDBMS, ParaLite, MySQL Cluster, Workflow applications

1. はじめに

1.1 大規模データ処理とそれを支えるシステム

近年，大規模クラスタ環境やクラウドシステムが一般的に利用可能になってきた．このような巨大な計算リソースを用い，巨大なデータを扱うアプリケーションを実行する例も多く見られるようなった．

大規模計算環境を利用した計算には，現在 Hadoop [1] に代表される MapReduce [8] 処理系が用いられることが多い．MapReduce により，既に多くの計算が大規模並列データ処理の恩恵を受けているが，map 関数や reduce 関数によって記述できる処理は限定的である．

また，データ管理の側面に着目した場合に，MapReduce 処理系で用いられる一般的なファイルベースのデータ管理では不十分なことが考えられる．例えば Hadoop では，HDFS (Hadoop Distributed File System) によって，ノードをまたがるデータ管理の透過性や，レプリケーションに

[1] 東京大学
University of Tokyo, 7-3-1 Hongo Bunkyo-ku, Tokyo 113–0033, Japan
[a)] nakatani@eidos.ic.i.u-tokyo.ac.jp
[b)] chenting@eidos.ic.i.u-tokyo.ac.jp
[c)] tau@eidos.ic.i.u-tokyo.ac.jp

よる冗長性を確保している．しかし，大規模データを扱う際には，多くのファイルが生成されてしまうことが一般的であり，それを統一的に管理し，データの読み書きを高性能に行うことは難しいと考えられる．ファイルを基盤としたデータ管理を一歩推し進め，統一性や効率性をサポートしたシステムとして，リレーショナルデータベース管理システム (RDBMS) が挙げられる．RDBMS では，処理すべきデータの構造をテーブルによって厳格に定義することができる．また，テーブルに対するデータの読み書きの性能については RDBMS が最適化を施している場合が多い．したがって，大規模データ処理の基盤として RDBMS を使用できれば，処理を記述するユーザがデータを統一的に扱え，データ処理の効率について多くを気にする手間からも解放されることが考えられる．

1.2 並列 RDBMS

RDBMS はデータ処理に役立つと考えられる特性を持つが，大規模な計算環境で使用ためするには，管理コストや拡張性の面で優れたシステムが求められる．この要求を満たすような，複数ノードでの運用を前提とする並列 RDBMS が存在する．多くの並列 RDBMS は，ノードの追加に対する性能向上 (スケールアウト性) や，高可用性を実現することを目指し実装されている．

商用の並列 RDBMS 製品としては，Oracle 社の Oracle RAC (Real Application Cluster) などが有名である．しかし商用製品は，システムに対する対価のみならずライセンス料やサポート料を負担する必要があることも少なくなく，万人が利用できるというわけではない．その点でオープンソースソフトウェアである並列 RDBMS の利用価値は高く，その例として，MySQL Cluster [2] や我々が開発する ParaLite [7], [6] などが存在する．本稿ではこの 2 つの並列 RDBMS に着目し，評価を行う．

1.3 ワークフローアプリケーションにおける RDBMS の役割

RDBMS が主に扱う処理として，OLTP (Online Transaction Processing) や DWH (Data Warehouse) が挙げられる．多くの RDBMS は，このような処理を高信頼かつ高性能に行えるように実装をされており，MySQL Cluster もこのような用途を想定している．

一方，ParaLite は OLTP や DWH のような処理よりも，ワークフロー処理のバックエンドとしての働きに重きを置いたシステムである．

OLTP や DWH といった処理は SQL クエリによって行われるが，ワークフロー処理はより高レベルなものと言える．ワークフローの記述は，何らかの処理のまとまりを担当するタスクと，タスク間の依存関係を定義することによって行われる．タスクとその依存関係を利用した記述には制約が少なく，自然言語処理や画像処理など，様々な処理をワークフローによって実行することが可能である．

ワークフローの基盤として RDBMS を使うことが考えられる．その際，タスクの行う一般的な処理は，
(1) データベースからデータを取得する．
(2) データに対し計算を行う．
(3) 計算を適用したデータをデータベースに書き出す．
といったものと考えられ，(1) - (3) を透過的なインターフェイスでサポートする機構がワークフロー基盤としての RDBMS には重要となる．後の章において，ParaLite はそのような機構を備えていることを指摘する．

また，(2) では必ずしも RDBMS の機能による処理が行われるわけではなく，外部コマンドの利用などが考えられる．しかし，(1) や (3) は RDBMS の速度が性能を左右するため，ワークフローを高速に動作させるには RDBMS 自身の性能も重要である．そこで我々は，ParaLite と MySQL Cluster の性能を [12] に見られるシンプルなベンチマークを使用して比較した．更に，実際にワークフローで記述されたアプリケーションの一つである格フレーム解析を一部 ParaLite で実行し，その性能を評価した．

1.4 本稿の構成

本稿では，オープンソースソフトウェアとして利用可能な並列 RDBMS である MySQL Cluster と ParaLite の性能を比較する．とりわけ，ワークフローアプリケーションの基盤としての重要な要件を満たしているかに着目する．

2 章では，本研究の関連研究を述べる．3 章では，ParaLite と MySQL Cluster の概要をそれぞれ述べ，それらの得意とする処理や，利用する際の構成について明らかにする．4 章では，シンプルな SQL クエリの処理性能を評価することによって 2 つのシステムの基本的な性能を比較すると共に，ワークフロー基盤として重要な機構を兼ね備えた ParaLite で格フレーム解析アプリケーションを一部動作させた際の性能を報告する．5 章で本研究をまとめ，今後の課題について述べる．

2. 関連研究

我々は，ワークフロー処理を得意とする並列 RDBMS として ParaLite を開発し，その特性や性能評価を公開してきた．その性能評価では MapReduce 処理系である Hadoop との比較を行っていたが，本論文では，比較対象としてオープンソースソフトウェアの並列 RDBMS である MySQL Cluster を用いた．これにより，並列 RDBMS の中での ParaLite の位置づけと， ParaLite の今後の課題を明確にすることを目的とする．

大規模データ処理をワークフローによって実行する研究は盛んに行われており，[9] がその一例として挙げられる．これらの研究においてデータの管理はファイルを単位とし

て行われており，そのコストは RDBMS の統一的なインターフェイスと高速な処理性能によって抑えられる可能性を残している．

MySQL Cluster の性能は，[11] や [14] などに報告されている．

[14] は，OLTP 処理のベンチマークである TPC-C [4] を MySQL Cluster で実行し，その性能について議論したものであり，適切なパフォーマンスチューニングを施せば，MySQL Cluster は TPC-C においてよいスケールアウト性能を発揮することを示している．

[11] は，MySQL Cluster で DWH 処理のベンチマークである TPC-H [5] を実行し，その性能について議論したものである．DWH 処理では一般に，1 つ 1 つのクエリの負荷が大きいものとなっている．そのため，SQL クエリを受け取って解析し，それを元にデータノードにデータ要求を行う SQL ノードが並列化されてなければ，1 つの SQL ノードに処理が集中するためにスケールアウト性能が発揮できない．MySQL Cluster 自身には SQL ノード間で 1 つのクエリを並列処理する機構が備わっていないため，この研究では vParNDB というミドルウェアを開発し，SQL ノードの並列化を実現し，スケールアウトを実現している．一方で ParaLite は，後述するように Collective Query という SQL クエリの処理を並列化する機構を有するため，DWH のように負荷の高いクエリ処理を並列実行できる．

本研究では， ParaLite を用いて実行するワークフローアプリケーションとして，格フレーム解析を用いた．格フレーム解析を，クラスタ環境用シェル GXP [10] を用いて大規模並列環境で行った研究として [13] があり，評価実験の参考とした．

3. 評価対象システムの概要

ここでは，評価の対象である ParaLite と MySQL Cluster の機能の概要を述べる．その中で，各システムが扱いを得意とする問題に関して述べる．特に，ParaLite は ワークフロー基盤として重要な機能を備えていることを明らかにする．

3.1 ParaLite の概要

ParaLite は，我々の開発する並列 RDBMS である．これは SQLite をベースにしたソフトウェアであり，RDBMS としての基本的な機能は SQLite に依っている．ParaLite は SQLite の基本機能に加え，分散環境で動作させるために以下のような機能を有する．

- SQL クエリを解析してジョブという単位に分割し，そのジョブを他の ParaLite プロセスに分散する．
- データをデータベースに格納するときに，テーブルのレコードを分散させる (Sharding)．

ParaLite を動作させるノードはディスクなどの資源を共有している必要はなく，いわゆる Shared-Nothing システムを想定している．耐故障性については，テーブルのレプリケーションをサポートしている．あるレコードにアクセスしようとしたときに，そのレコードを持つ 1 つのノードが到達不能になった場合でも，そのレコードのレプリケーションを持つ別のノードが到達可能であれば，レコードへアクセスすることが可能である．

ParaLite は，次の 3 つの構成要素から成るシステムである．

マスタノード クライアントから SQL クエリを受け取り，クエリ解析を行いジョブへと分割し，ジョブを各クライアントに送り返す．ワーカノードのデータ転送終了通知の受け取り，クライアントが通信すべきデータノードの選択など，システム全体の調整も行う．マスタノードはクライアントの中から選定される．

クライアント SQL クエリを発行し，それをマスタノードへ送る．その結果，マスタノードから返されたジョブに従い，処理を実行する．ジョブの処理の過程で，マスタノードから指示されたデータノードとデータを受送信する．

データノード データをテーブルから読み書きし，クライアントとデータをやり取りする．

クライアントが複数存在し，それらが同一のクエリを発行したとき，マスタノードはそのクエリ解析結果のからジョブを複数クライアントに分散させる．これを **Collective Query** と呼んでおり，ParaLite は Collective Query によって SQL クエリの並列実行をサポートしている．

ParaLite をワークフローアプリケーションの基盤として扱う際に重要な機能として，**UDX (User-Defined eXecutables)** がある．これは，クライアントが発行する SQL クエリの中にシェルコマンドを組み込める機能である．以下に UDX を用いた SQL クエリの例を示す．

```
SELECT F(contents) FROM document_table
WITH
  F="text_procession_prog"
```

これは，

(1) document_table テーブルから contents カラムのデータを取り出し，
(2) そのデータを標準入力から text_procession_prog プログラムに渡し，
(3) text_procession_prog の出力結果を標準出力に表示する．

という働きのクエリである．UDX により，データをデータベースから取り出し，データに対して何らかの処理を適用し，その結果を得るという，ワークフローの各タスクが行うべき基本的な操作が自然な形で記述できている．また，ParaLite は通常の RDBMS と同様にサブクエリを扱うこ

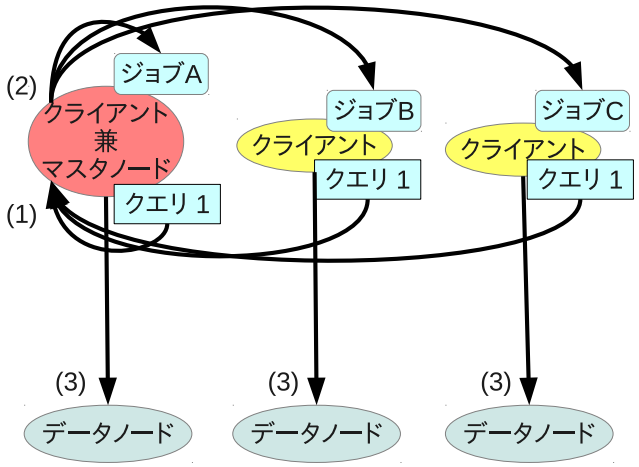

図 1 ParaLite のアーキテクチャ
Fig. 1 ParaLite Architecture

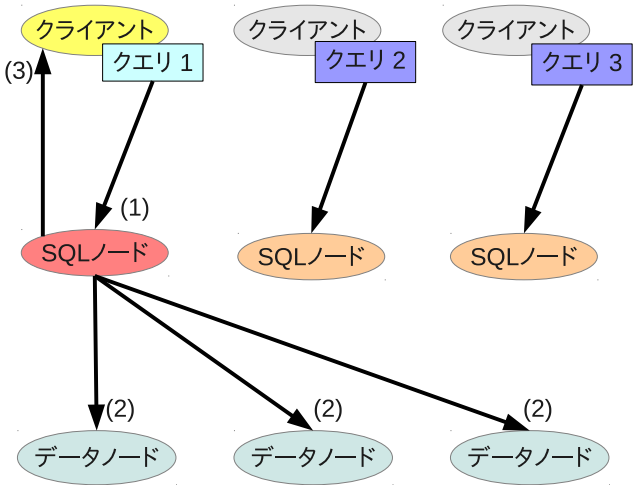

図 2 MySQL Cluster のアーキテクチャ
Fig. 2 MySQL Cluster Architecture

とができ，それによって UDX のシェルコマンドの出力結果をそのまま新たなテーブルに格納することもできる．

```
CREATE TABLE processed_document_table
AS
SELECT F(contents) FROM document_table
WITH
  F="text_procession_prog"
```

この例では，text_procession_prog の出力結果が processed_document_table に格納されることとなり，ワークフロー中で後に実行されるタスクが processed_document_table を利用することができる．

以上を踏まえ，ParaLite の動作を図 1 によって説明する．図には見やすさを考慮し手順 (3) までのみ描かれている．

(1) 複数のクライアントが同一のクエリ (Collective Query) を発行する．このクエリは，マスタノードに送られる．
(2) マスタノードがクエリ解析をし，ジョブに分割する．ジョブは Collective Query を発行したクライアントに分散される．
(3) 各クライアントがジョブに従いデータノードとデータ通信を行う．
(4) Collective Query に UDX が含まれていれば，クライアントはデータノードから取得したデータに対して UDX を適用する．この仕組みにより，UDX は並列実行される．
(5) 各クライアントが処理結果をマスタノードに通知する．

UDX と類似のものとして，UDF (User-Defined Function) がある．これは，SQL 中でデータに適用可能な関数を，ユーザが独自にプラグインとして定義できる機能で，MySQL を始めとする多くの RDBMS でサポートされている．しかし，UDF を作成するには，各 RDBMS の定めるインターフェイスに従って処理を記述し，それを共有ライブラリとしてコンパイルし，UDF 使用前にその共有ライブラリを読み込むといった手順をユーザが踏む必要があり，UDX と比べるとその使用は困難であると言える．

ワークフロー中には依存関係がなく並列に実行できるタスクが現れることがあり，それに対しては Collective Query による並列実行が可能である．また，UDX によるシェルコマンドの活用は，多数の処理から構成されるワークフローに対しては非常に効果的である．以上の点から，ParaLite はワークフロー基盤として適した並列 RDBMS であるということができる．

3.2 MySQL Cluster の概要

MySQL Cluster は Oracle 社の提供するオープンソースソフトウェアの並列 RDBMS である．欧米においては既に通信キャリアなどでの稼働実績があり，国内においても今後導入が進むことが予測される．

ParaLite と同様，MySQL Cluster も Shared-Nothing システムで動作する．耐故障性に関する機能は充実しており，2 フェーズコミットによるレプリケーションやクエリログによる障害からの復帰などをサポートしている．実装として MySQL の大部分を利用できているので，耐故障性対策についても長年の実績のある MySQL の成果を利用できている．

MySQL Cluster の 3 つの構成要素を以下に挙げる．

クライアント SQL クエリを発行し，それを SQL ノードへ送り，処理結果を SQL ノードから受け取る．
SQL ノード SQL クエリを受け取り，クエリ解析を行い，解析結果に行い処理を実行する．処理の過程で，データノードとデータを受送信する．
データノード データをテーブルから読み書きし，クライアントとデータをやり取りする．

上述の構成要素が協調して動作する様子を，図 2 により

説明する．

(1) クライアントがクエリを発行し，SQL ノードへ送る．この際，1 つクエリを複数の SQL ノードで処理するような機構は MySQL Cluster に通常備わっておらず，1 つのクライアントは 1 つの SQL ノードと通信をするようなモデルとなっている．これは 1 つのクエリを並列実行する Collective Query とは異なるが，複数のクライアントが別々のクエリを発行するときに，各クライアントは別の SQL ノードを選定してクエリ解析をさせることで処理の分散が可能であることに注意．

(2) SQL クエリを受け取った SQL ノードがクエリ解析をし，データノードと通信する．

(3) SQL ノードが処理結果をマスタノードに通知する．

MySQL Cluster では SQL ノードやデータノードを増やすことでスケールアウトを図ることができ，実際に複数の性能評価によってそれが示されている．その一方で，SQL ノードが複数ある時でも，1 つ 1 つのクエリに対しての処理の分散はサポートされていない．SQL ノードを増やす目的は，OLTP のような処理において多数のクライアントからのトランザクション要求に耐えるためであることが一般的である．DWH のように 1 つ 1 つのクエリが高負荷な処理を並列に処理したい場合には，vParNDB ミドルウェア [11] のような特別な拡張が必要となる．また，ワークフローへの対応を考えたとき，ParaLite の Collective Query の説明において述べたように，ワークフローには並列実行可能なタスクが存在する．それらのタスクが同一の SQL クエリによりデータを習得するようなものであった場合，SQL ノードが並列化をサポートしていないため，実際には並列実行をすることができない．

更に，今回の調査の範囲で，MySQL ではサポートされている UDF が，MySQL Cluster ではサポートされていないことが分かった (付録参照)．

4. 評価

4.1 評価環境

評価には，次の同構成のマシンを複数台用いた．

CPU Intel Xeon CPU E5530 @ 2.40GHz, 物理 8 コア, Hyper-Threading.

メモリ 容量 24 GB

ローカルディスク 容量 500 GB, キャッシュ 32 MB, SATA 3Gbps.

NFS ディスク 10 本で構成される RAID6. 4Gbps のファイバーチャネルで接続．1 ディスク当たり，容量 1TB, キャッシュ 32 MB.

ネットワーク帯域 10 Gbps

4.2 シンプルな SQL クエリの処理性能

MySQL Cluster と ParaLite の基本的な性能を調査するために，[12] に記載された Selection Task のベンチマークを実行した．

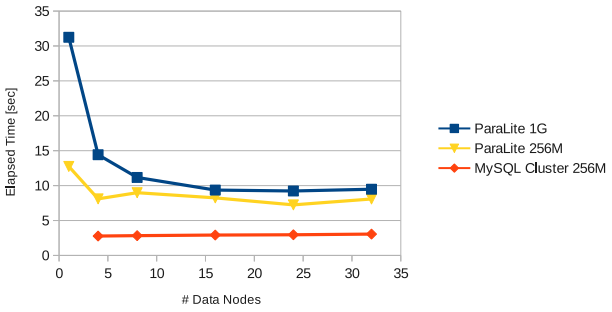

図 3 Selection Task ベンチマークの実行性能
Fig. 3 Performance of Selection Task benchmark

Selection Task ベンチマークでは，次のテーブル定義を用いた．

```
CREATE TABLE Rankings (pageRank INT, pageURL
    VARCHAR(100), avgDuration INT);
```

このテーブルに対しあるサイズのデータを格納し，次のクエリの応答時間を計測した．

```
SELECT pageURL, pageRank FROM Rankings WHERE
    pageRank > 3;
```

pageRank カラムにはインデックスが付与されていないため，全件検索のクエリとなる．このクエリを，入力データ 1G バイトの ParaLite, 入力データ 256M バイトの ParaLite, 入力データ 256M バイトの MySQL Cluster で比較した結果を図 3 に示す．

入力データ 1G バイトの ParaLite の結果より，データノード数を増加させるほど，クエリの処理時間が短くなる傾向が見て取れる．これは，データノード数が増えるほど，1 つのデータノードの持つデータサイズは小さくなり，結果として各データノードにおける処理時間が短くなったためと考えられる．しかし，データノード数が増えすぎると，実行時間はそれ以上短くならないことも確認できる．この原因は，各データノードの持つデータ量が小さくなるにつれ，各データノードでのジョブの処理よりも，ネットワーク遅延のオーバヘッドが効いてくるためであると考えられる．今回の計測対象外ではあるが，MySQL Cluster も同様の傾向を持つことが予想される．

入力データ 256M バイトの ParaLite と MySQL Cluster の結果を見ると，ともにデータノード数の増加による性能向上はほぼ見られない．これも上述のように，各データノードのデータサイズが小さく，ネットワーク遅延のオーバヘッドが掛かっているためと考えられる．また，ParaLite の計算時間は MySQL Cluster の計算時間よりも長くなっている．この差異は，ParaLite と MySQL Cluster のデータノードの実装の違いから来るものである可能性が考えら

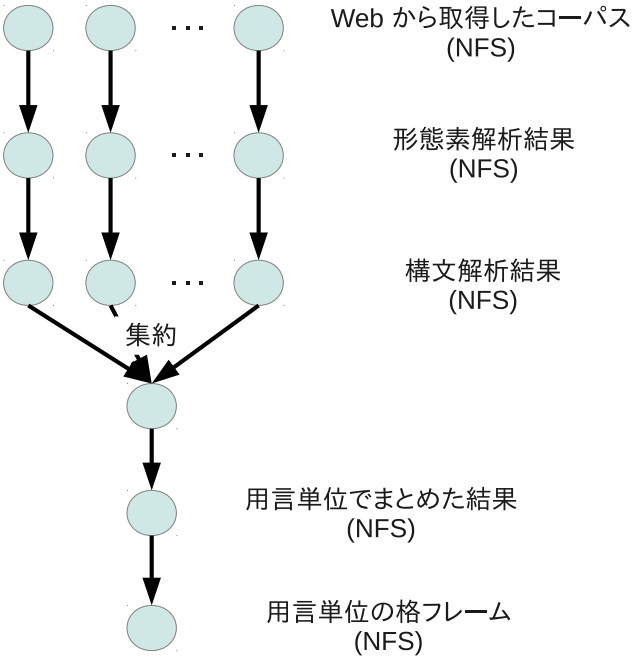

図 4 格フレーム解析ワークフロー (ファイルベース手法)
Fig. 4 Case Frame Construction workflow (File-based approach)

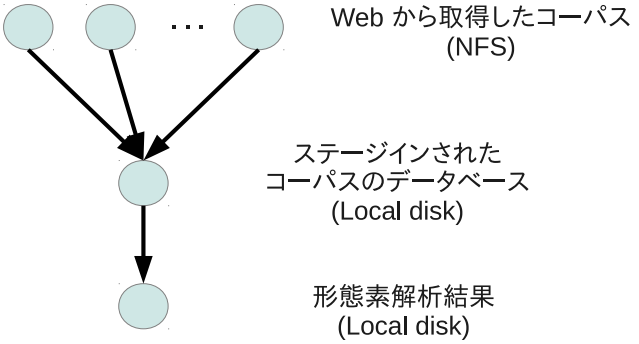

図 5 格フレーム解析ワークフローの一部 (ParaLite を基盤とした手法)
Fig. 5 Part of Case Frame Construction workflow (ParaLite-based approach)

れる．ParaLite のデータノードの実装には，SQLite のライブラリを使用しており，これが十分に高速でない可能性や，ParaLite 自身のコードが低速である可能性があり，今後このボトルネックを解析し解消する必要がある．

4.3 格フレーム解析ワークフロー

ParaLite は UDX と Collective Query により，外部プログラムを使用するようなワークフローの並列実行をサポートする．MySQL Cluster は，今回の調査の範囲では， UDF とその並列実行ををサポートしていないため，MySQL Cluster でのワークフローの実行は見送った．

今回の評価では，ワークフローアプリケーションとして格フレーム解析を一部実行した (図 4, 5)．格フレームは，

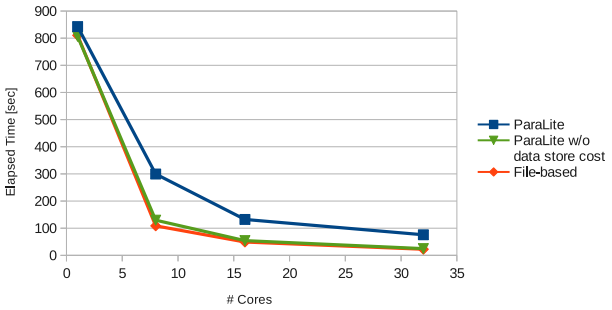

図 6 格フレーム解析ワークフローの実行時間
Fig. 6 Execution time of Case Frame Construction workflow

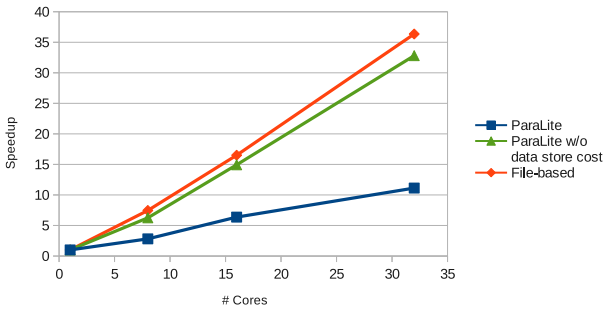

図 7 格フレーム解析ワークフローの台数効果
Fig. 7 Speedup of Case Frame Construction workflow

用言とそれに関係した名詞を集めたものであり，一定のコーパスに対し格フレーム解析を施すことで，検索，要約，翻訳といった自然言語処理アプリケーションに応用可能なデータを得ることができる．本評価では，この内のコーパスから形態素解析結果を得るステップを実行した．

ParaLite の比較対象として，ファイルベースな手法を用いた．図 4 は，ファイルベース手法を念頭に置いたものである．Web から取得したコーパスを NFS に配置した入力とし，それを形態素解析すると共にローカルディスクへステージインする．図 5 は，ParaLite による実行の流れを示したものである．NFS 上のコーパスをローカルディスク上のデータベースに保存する．データベースはユーザからは 1 つに見えるが，実際には ParaLite が複数のデータノードに分散している．

入力コーパスのサイズは合計で 323M バイトであり，構文解析結果は合計で 5.2G バイトとなった．

構文解析の並列化手法について，ファイルベース手法では，構文解析のタスクを各コアに割り当て並列実行を行う．タスク数は今回の評価で用いたコア数よりも十分に多くある．ParaLite では，分散されたデータベースに各ノードに 1 つだけ存在するデータノードがアクセスし，コーパスをデータベースから読み取り構文解析結果をデータベースに格納する．クライアントは各ノードに最大でコア数と同

じ 8 個立ち上げる．つまり，クライアントを 16 個立ち上げた際は，2 ノードを使用し，クライアントは 2 つのデータノードと通信を行う．

実行時間計測は，ファイルベース手法ではステージインから構文解析結果を出力するまでを測り，ParaLite ではデータベースからコーパスを得てデータベースに構文結果を格納するまでを測定した．

実験結果を図 6, 7 に示す．得られた結果を考察する．

まず，ParaLite ではクライアントの数を増やすにつれて，ワークフローの実行時間を短くすることができた．だが，ParaLite はファイルベースに比べて実行時間で差を付けられており，台数効果も十分に発揮できているとは言えない．この原因を解析したところ，形態素解析を施したデータをデータベースに格納する際の時間が主なボトルネックとなっていた．そこで，データベースへの格納の時間を差し引くと，ファイルベースとほぼ同様な実行時間と台数効果になることが確認できた．

しかし，ファイルベースの実行時間には， NFS から入力ファイルをメモリに読み込むコストと，形態素解析プログラムの出力をファイルに書き出すコストが含まれているのにもかかわらず，ファイルベースの性能は，データベースへの格納時間を差し引いた ParaLite よりも良い．これは，次の 2 つの理由から説明できる．

(1) 形態素解析プログラムは CPU インテンシブであり，データの読込時間は計算の時間に対して無視することができた．
(2) ParaLite の方がファイルベースに比べて並列度を発揮するのに時間を要していた．すなわち，ParaLite が UDX を並列に実行するには，クライアントがマスタノードに登録をし，マスタノードからジョブを受け取る必要があるが，全てのクライアントがジョブを受け取るまでにタイムラグがあり，その間は並列度がクライアントの数だけ発揮されていなかった．

また，特にファイルベースにおいて台数効果が線形よりも良くなっているのは，形態素解析プログラムの計算量がデータ量よりもオーダの上で大きいため，コア数が大きくなり，各形態素解析プログラムが扱うデータが小さくなるにつれ，その実行時間が線形以上に短くなったものと考えられる．

この考察から，ParaLite の性能上の問題が次のように挙げられる．

(1) 各クライアントが UDX の処理の結果をデータノードに格納する際のオーバヘッドが大きい．これは実装上の問題であり，十分な台数効果を発揮するためには改良する必要がある．
(2) Collective Query を並列実行する際に，各クライアントが実行を開始する時間にラグが存在する．ただし，これはクライアントが扱う処理の時間が長くなるにつれ無視できるものとなる．

5. 結論

本研究では，大規模データ処理を記述できるワークフローと，その基盤として有望な並列 RDBMS に着目し，並列 RDBMS である ParaLite と MySQL Cluster の性能評価を行った．

ParaLite と MySQL の比較ではシンプルな SQL クエリを用い，並列 RDBMS が処理を分散できることと，ParaLite の基礎的な性能は MySQL Cluster よりも現状で低いことを示した．

一方で，ParaLite にはワークフローアプリケーション基盤として重要と考えられる Collective Query と UDX の機構があることについて触れ，実際にそれを用いて格フレーム解析アプリケーションを一部動作させた．結果はファイルベースなワークフローよりも劣るものとなったが，ParaLite の実装上のボトルネックを指摘し，それを解消すればファイルベースのワークフローに性能面で追いつける可能性があることを述べた．

今後の課題について述べる．まず，ParaLite のワークフローの記述性を詳細に検討する必要がある．これは特にファイルベースの手法と比較する必要があるが，他の RDBMS や MapReduce 処理系との比較も有用であると考えられる．また，ParaLite には多くの性能上の改善点がある．それには ParaLite 自身の実装による影響もあり，SQLite の実装から来る影響も考えられる．現在のボトルネックを詳細に解析し，性能向上に努めて行きたい．

謝辞 本研究の一部は，総務省委託研究「広域災害対応型クラウド基盤構築に向けた研究開発 (高信頼クラウドサービス制御基盤技術)」において実施された．

参考文献

[1] Apache Software Foundation. Hadoop.
[2] MySQL Cluster. http://www.mysql.com/products/cluster/.
[3] MySQL Documentation: MySQL Reference Manuals. http://dev.mysql.com/doc/.
[4] TPC Benchmark C. http://www.tpc.org/tpcc/.
[5] TPC Benchmark H. http://www.tpc.org/tpch/.
[6] Ting Chen and Kenjiro Taura. Supporting Collective Queries in Database System for Data-intensive Workflow Applications. **情報処理学会研究報告**. *[ハイパフォーマンスコンピューティング]*, 2011(28):1–8, 2011-07-20.
[7] Ting Chen and Kenjiro Taura. ParaLite: Supporting Collective Queries in Database System to Parallelize User-Defined Executable. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:474–481, 2012.
[8] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
[9] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P. Berman, and Phil Maechling. Data Sharing Options for Scientific

Workflows on Amazon EC2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–9, Washington, DC, USA, 2010. IEEE Computer Society.

[10] Kenji Kaneda, Kenjiro Taura, and Akinori Yonezawa. Virtual Private Grid: A Command Shell for Utilizing Hundreds of Machines Efficiently. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:212, 2002.

[11] Sudsanguan Ngamsuriyaroj and Rangsan Pornpattana. Performance Evaluation of TPC-H Queries on MySQL Cluster. 0:1035–1040, 2010.

[12] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[13] 河原 大輔 and 黒橋 禎夫. 高性能計算環境を用いた Web からの大規模格フレーム構築. **情報処理学会研究報告**. **自然言語処理研究会報告**, 2006(1):67–73, 2006-01-12.

[14] 中山 陽太郎 and 林 宏祥. MySQL Cluster によるクラスタ・データベースの実装と評価 (特集データエンジニアリング). *Unisys* **技報**, 29(2):183–193, 2009-08.

付　　録

A.1 MySQL Cluster が UDF をサポートしていないとする根拠

MySQL Cluster の概要において，UDF のサポートがされていないという報告をした．ここでは，その根拠について記述する．

(1) MySQL Cluster のドキュメント [3] に UDF について一切記載がなかった．

(2) シンプルな UDF を作成して使用した結果，MySQL では正しく動作し，MySQL Cluster では誤動作をした．作成した UDF は，ある文字列型のカラムについて，レコードのデータをそのまま返す関数であったが，MySQL Cluster では返された文字列の末尾に余分な文字が混ざっていた．MySQL で動作させる場合にはストレージエンジンに InnoDB を使用した．MySQL Cluster ではストレージエンジンに NDB を使用し，データノードを 2 台設置した．

以上の点から，MySQL Cluster は UDF をサポートしていないと判断した．